CS 362, HW 8

Prof. Jared Saia, University of New Mexico

1. Do Problem 4 on the midterm. Also, use your recurrence relation to fill in a table for the input cost array $[2, 1, 5, 8, 6, 2]$.

2. Do Problem 5 on the midterm. Also, use your recurrence relation to fill in a table for initial input $(2, 2, 2)$ of pile sizes. Your table should be 3 dimensional so please describe it via 2-D slices. For example if your recurrence is $f(x, y, z)$. Then give the three 2-D tables $f(0, y, z)$, $f(1, y, z)$ and $f(2, y, z)$, for all values y and z. Can the first player force a win if the piles are of sizes $(2, 2, 2)$?

3. Consider the following alternative greedy algorithms for the activity selection problem discussed in class. For each algorithm, either prove or disprove that it constructs an optimal schedule.

   (a) Choose an activity with shortest duration, discard all conflicting activities and recurse

   (b) Choose an activity that starts first, discard all conflicting activities and recurse

   (c) Choose an activity that ends latest, discard all conflicting activities and recurse

   (d) Choose an activity that conflicts with the fewest other activities, discard all conflicting activities and recurse

4. Now consider a weighted version of the activity selection problem. Imagine that each activity, a_i has a *weight*, $w(a_i)$, and weights are totally unrelated to activity duration. Your goal is now to choose a set of non-conflicting activities that give you the largest possible sum of weights, given an array of start times, end times, and values as input.

   (a) Prove that the greedy algorithm described in class - Choose the activity that ends first and recurse - does not always return an optimal schedule for this problem

(b) Describe an algorithm to compute the optimal schedule in $O(n^2)$ time. Hint: 1) Sort the activities by finish times. 2) Let $m(j)$ be the maximum weight achievable from activities $a_1, a_2, \ldots, a_j$. 3) Come up with a recurrence relation for $m(j)$ and use dynamic programming. Hint 2: In the recursion in step 3, it'll help if you precompute for each job j, the value x_j which is the largest index i less than j such that job i is compatible with job j. Then when computing $m(j)$, consider that the optimal schedule could either include job j or not include job j.